# 五子棋实验报告

## 汇报人：计算机科学与工程学院

## 计算机类12班

## 温馨

## 2023080912027

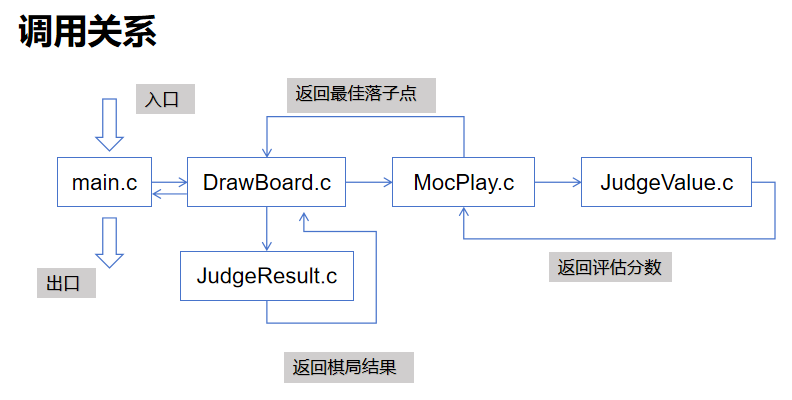
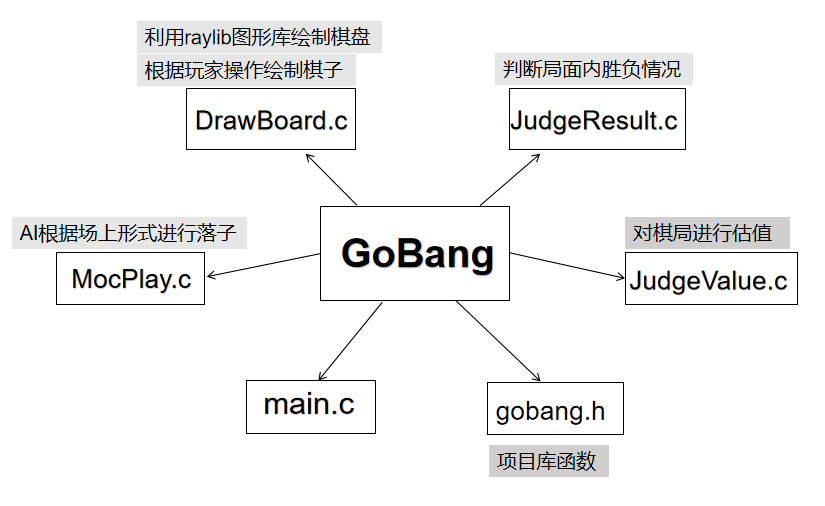
# 实验目的

编程实现可进行人机对战的五子棋AI。

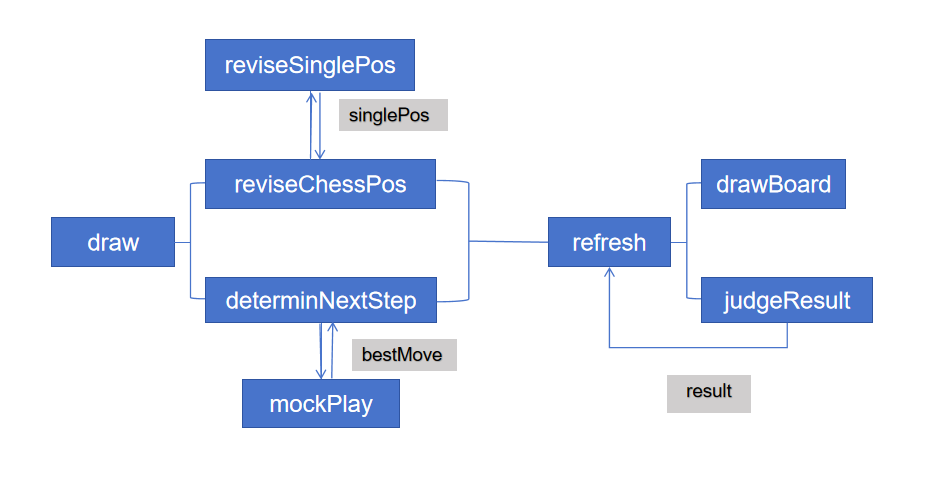
# 实现要求

1. 利用raylib图形库绘制棋盘。
2. 根据玩家操作绘制棋子。
3. AI根据场上局势进行落子。
4. 判断局面内胜负情况，并输出对局结果。

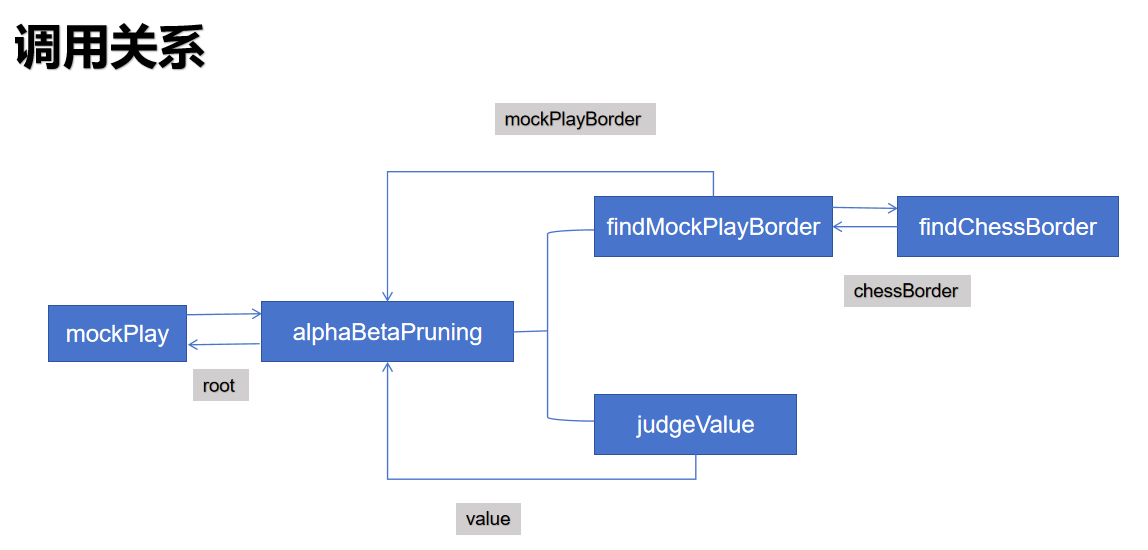
# 实现方式（V1.0）



### DrawBoard.c



### MockPlay.c



## 利用raylib图形库绘制棋盘。

### 初始化窗口；

* 定义InitialWindow函数，
* *调用InitWindow函数，设置窗口尺寸和名字。*

### 记录棋盘相关的数据

* 创建数组borderLine，元素为Rectangle结构变量，记录棋盘边界位置；
* 创建数组benchMark，元素为Rectangle结构变量，记录棋盘中的基准点位置。

### 根据相关数据绘制棋盘

* 定义drawBoard函数，实现绘制棋盘的功能。
* *调用ClearBackground函数，设置窗口背景颜色为棋盘黄*；
* *调用DrawRectangleRec函数，传入borderLine，for循环迭代绘制棋盘边界线条；*
* *创建Rectangle结构变量XInnerLine和YInnerLine，初始化为棋盘中第一条横竖内线的位置；*
* *调用DrawRectangleRec函数，传入XInnerLine和YInnerLine，for循环迭代绘制棋盘内线。每一次迭代均为XInnerLine的纵坐标和YInnerLine的横坐标加上棋格的边长来实现用单个变量在不同位置绘制棋盘内线；*
* *调用DrawRectangleRec函数，传入benchMark，for循环迭代绘制基准点。*

## 根据玩家操作绘制棋子。**附图一、三**

### 记录棋子的位置

* 创建数组chessPos，元素为Vector2类型，记录棋子的屏幕坐标；
* 创建数组chessPoint，元素为int类型，记录棋子的棋盘坐标；
* 定义变量chessCount，初始化为0，记录下子个数。

### 读取玩家的操作

* 调用IsMouseButtonPressed(MOUSE\_LEFT\_BUTTON)判断鼠标是否有点击操作；
* 调用GetMousePosition函数获取鼠标点击的位置。

### 处理玩家点击位置，使棋子的绘制位置恰好落在格点上

* 定义reviseChessPos函数，接收鼠标的点击位置（Vector2类型），返回修正后的坐标（Vector2类型）；
* *调用reviseSinglePos函数，依次对x、y进行修正，而后返回。*
* *将MultiplesOfUnitLength赋值给MultiplesOfUnitLengthX或MultiplesOfUnitLengthY。*
* 定义reviseSinglePos函数，接收单个原始坐标（float），返回修正后的单个坐标（float）。（额外定义一个单个修正函数是为了减少重复代码量）**附图四**
* *定义变量margin，存放棋盘的边距；*
* *定义变量unitLength，存放棋盘格点间单位长度；*
* *定义变量MultiplesOfUnitLength，初始化为0，记录点击位置走过的单位长度个数；定义变量MultiplesOfUnitLengthX，记录行单位偏移量；定义变量MultiplesOfUnitLengthY，记录列单位偏移量。这三个变量都为全局变量，便于根据行列单位偏移量记录棋子的棋盘坐标；*
* *为接收的原始坐标减去margin，便于后续确定其相对于单位长度的偏移量；*
* *使用while循环进行迭代，每次迭代给原始坐标减去一个单位长度，并为MultiplesOfUnitLength加一，直到原始坐标的值小于一个单位长度为止；*
* *判断此时的原始坐标和棋格半边长的大小关系，如果小于半边长，则舍去；若大于，则给MultiplesOfUnitLength加一。这一步操作用于处理鼠标点击位置不在格点上的情况。*
* *返回margin+MultiplesOfUnitLength\*unitlength，这样就保证了棋子的位置在格点上。*

### 判断鼠标点击位置是否和已存在的棋子位置重合

* 利用for循环，将reviseChessPos的返回值与chessPos的内容逐个比较，当前者与后者的每一个元素都不相同时，将其录入chessPos的第chessCount个元素，以及chessPoint的【MultiplesOfUnitLengthY】【MultiplesOfUnitLengthX】位置。

### 绘制棋子

* 调用DrawCircleV函数，for循环依次传入chessPos内的元素，当chessCount为奇数时，绘制白棋（AI）；当chessCount为偶数时，绘制黑棋（玩家）。

## AI根据场上局势进行落子。

### 记录AI落子位置

定义Vector2变量nextStep，记录落子位置

### 搜索局面中AI可能的落子位置并确定最终的落子位置

* 定义结构Border，结构成员top、bottom、left、right；
* 定义结构MockPlayBorder，结构成员beginLine、endLine、beginRow、endRow；
* 定义函数findChessBorder，返回Border结构变量。
* *定义Border结构变量border，初始化为0，用以记录当前棋盘下子的边缘行列位置。*
* *定义变量change，初始化为1，用以确认外层循环变量的更改值。*
* *使用双层循环来寻找棋盘的上下落子边界。进入外层循环后，首先判断border.bottom，如果bottom已经被赋值，直接退出循环。仅当其未被赋值时，进入内层循环。内层循环中，逐个检查chessPoint的值，是否为黑棋或白棋，若是，则利用change的值判断top是否被赋值，若change为1，则top未被赋值。此时将外层的循环变量赋值给top，更改外层循环变量为14，并将change赋值为-1，退出内层循环。下一次进入内层循环并找到落子点时，将外层循环变量赋值给bottom。*
* *使用双层循环来寻找棋盘的左右落子边界。外层循环变量从border.top开始，到border.bottom结束。外层循环中定义变量left和right，初始化为0，记录每一行中左右边界被检查与否。进入内层循环，逐个检查chessPoint直到找到落子点。首先判断left是否为0，若是，比较内层循环与border.left的值，若前者更小，则赋值给后者；若前者更大，则不改变border.left的值，将left赋值为1，内层循环变量赋值为14，进入下一次循环。右侧的赋值过程同上，不再赘述。*
* 定义函数findMockBorder，接收Border结构变量，返回mockPlayBorder结构变量。这样做是为了减少无效落子，将模拟落子的范围限制在已有落子范围的两格之内，减少搜索时间。
* *定义mockPlayBorder结构变量，初始化为0，记录模拟落子的边界。*
* *利用三目运算符，若border.top - 2小于0，即外拓两个之后超过了棋盘的边界，给mockborder.beginLine赋值为0；若否，则赋值为border.top - 2。其他三个方向的边界赋值同上，不再赘述。*
* 定义函数alphaBetaPruning，传入int类型的参数alpha，beta和depth，当depth为奇数时，返回beta；depth为偶数时，返回alpha。利用这个函数进行alpha-beta剪枝来确定落子位置。**附图五**
* *定义结构体Leaf，结构成员为x、y，记录落子的位置*
* *定义局部整型变量score，接收棋盘分值。*
* *定义局部结构变量root，存放根节点位置。*
* *定义全局结构变量bestMove，存放最佳落子位置。*
* *首先判断depth是否为0，即是否搜到了最底层。若是，调用judgeValue函数对当前整个棋局进行评分，并返回评分值。*
* *使用双层循环寻找chessPoint的未落子位。进入外层循环，首先判断alpha是否大于beta，若是，直接退出循环。若不是，进入内层循环，根据depth的值落子一次，将落子位置赋值给root，奇数落黑子（玩家），偶数落白子（AI），相应录入chessPoint。而后获取alphaBetaPruning函数接收alpha、beta和depth-1的返回值，赋给score。递归完成后，将落子位置设置为0，即撤销落子，而后根据depth的值进行判断。若depth为偶数，即AI层，当alpha小于score时，后者赋值给前者，且若此时正好在最顶层，则将root的值赋给bestMove。若depth为奇数，即玩家层，当beta大于score时，后者赋值给前者。最后判断alpha是否大于等于beta，若是，则退出内层循环。*

### 为每一个局面打分

* 定义枚举Score，存储各种棋型对应的分数。
* 定义函数judgeValueAll，返回整局棋的总评分。
* *定义局部变量value，初始化为0，存储棋局的总评分。*
* *双层循环找到chessPoint中每一个落子位置，调用FourDirectionIdentify对每一个棋子的四个方向进行评分，并加到value中。*
* *返回value。*
* 定义函数FourDirectionIdentify，接收目标棋子的行列坐标及判断方向，返回棋局的评分。**附图二**
* *定义局部变量chessNum, space, block, singleSideBlock, chessColor，分别记录棋子数、空格、异色子数、单边是否出现异色子、当前棋子颜色。*
* *循环遍历棋子该方向两侧五个位置的物体。设置循环变量line、row。初始化为0，每次循环均执行自加。进入循环，首先判断二者是否大于4，若是，则赋值为-（line或row - 4），即换向检查。*
* *switch语句根据输入的方向确定之后进行判断的循环变量的值。若水平，行变量line设置为0；若竖直，列变量row设置为0。若右下，不进行更改；若左下，始终将row设置为line的相反数。*
* *判断传入的posLine和posRow在加减line和row后是否超棋盘边界，若是，检查当前是否为反向判断，若是，直接退出循环；若否，将line赋值为4，进行下一次循环。*
* *switch语句逐个检查chessPoint的posLine+line，posRow+row位置，若为同色棋，chessNum加1；若为异色棋，block加1；若为空格，space加1。当单侧出现一个异色棋或两个空格后，换到异侧检查。换侧操作同上，不再赘述。*
* *全部检查完后调用judgeValueSingle函数，传入chessNum、space、block，返回该函数的返回值。*
* 定义函数judgeValueSingle，接收棋型中棋子数、空格数、异色子数，返回单个棋子单个判别方向的评分。

### 绘制AI落子

* 将bestMove的值赋给nextStep。
* 调用reviseChessPos函数，并将其参数choice设置为2，即调整机器走子位置。与调整玩家走子位置不同的是，这里的multiplesOfUnitLengthX和multiplesOfUnitLengthY直接为nextStep的x、y值。
* 其他绘制操作同玩家走子绘制，不再赘述。

## 判断局面内胜负情况，并输出对局结果。

* main函数中定义变量result，接收judgeValue函数的返回值，根据返回值调用DrawText函数，绘制不同的对局结果提示语句。
* *result=1时，绘制“You Win！”*
* *result=2时，绘制“You Lose！”*
* *result=3时，调用WindowShouldClose函数，判断玩家是否点击右上角退出键。若否，继续对局。*
* 定义judgeResult函数，返回1、2、3，分别代表胜利、失败、继续对局。
* *调用judgeValueAll函数，返回当前局面总分，并与宏中定义的WinScore、LoseScore进行比较，做相应返回。*

## 5.宏列表

BorderThickness 边界线宽度  
InnerLineThickness 内部线条宽度  
SpaceSideLength 棋格边长  
BorderSpace 空白边界宽度  
BenchMarkSideLength 基准点边长  
BoardSideLength 棋盘宽度  
CenterBenchMarkPos 中心基准点坐标;  
Interval中心基准点和其他基准点的间隔距离  
ChessRadius棋子半径  
ScreenWidth 窗口尺寸  
ScreenHeight  
BoardColor CLITERAL 棋盘背景色  
WinScore 胜利分数  
LoseScore 失败分数  
BlackChess 黑子  
WhiteChess 白子

# 四、后续优化与调整

## judgeResult函数

#### 对整局进行评分->定义一个bool类型变量isFive，对单个棋子的任意方向进行棋型判断和评分，当返回分数为FIVE时令isFive=1，再根据当前执子人返回胜利或失败。这样可以避免一方连五一方活四导致整局分数未达到Winscore的情况，并且可以减少循环检查的次数。

## 棋盘绘制

#### Main函数内beginDraw和endDraw->单独设置一个函数refresh，每次棋盘数组发生变化的时候进行绘制，就可以实时显示落子情况，无需等到AI算出结果后再显示玩家落子。并且加宽边缘空白，在空白位置显示“AI is thinking...”提示语句。

## 棋子搜索顺序

#### 从已下子边界开始搜索->从上一次下子的周围开始搜索，这样更符合对弈的一般规律。

## 库文件

#### 增加了对方向的宏定义，使代码更清晰。

# 五、运行时遇到的问题

#### 1.新落子可覆盖旧落子

##### 在把落子录入数组前检查是否有重合。

#### 2.在棋盘边线上的落子没有处于格点位置

##### 额外处理边线落子的坐标。

#### 3.棋型判别函数在实际运行时出了很多的问题，花了比较多的时间逐步调试才一一解决，不过也导致最终这个函数的逻辑比较复杂。

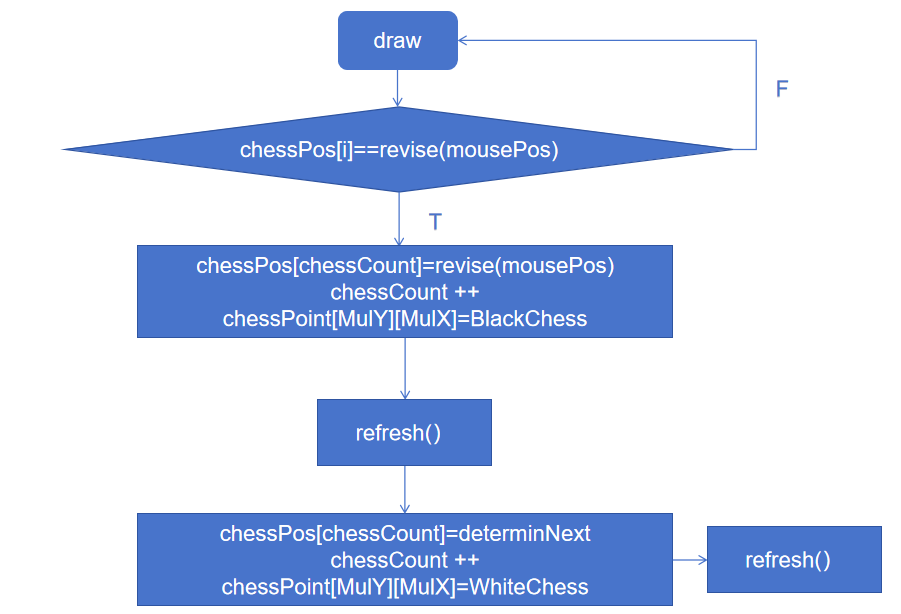
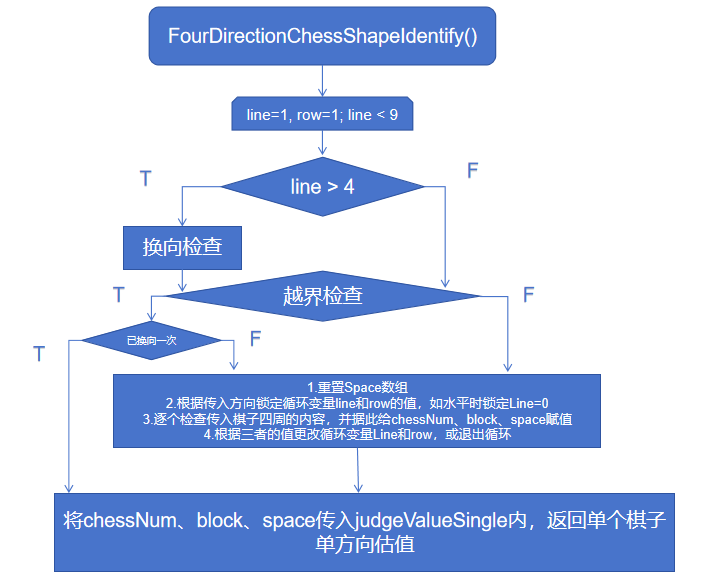
#### 4.最初的估值函数里面没有包含活一等棋型，后来自己一一加了上去。（因为这个估值函数的思路基本上照搬了参考资料里面的估值函数，但是他的棋型判别似乎跟我的思路有所不同）

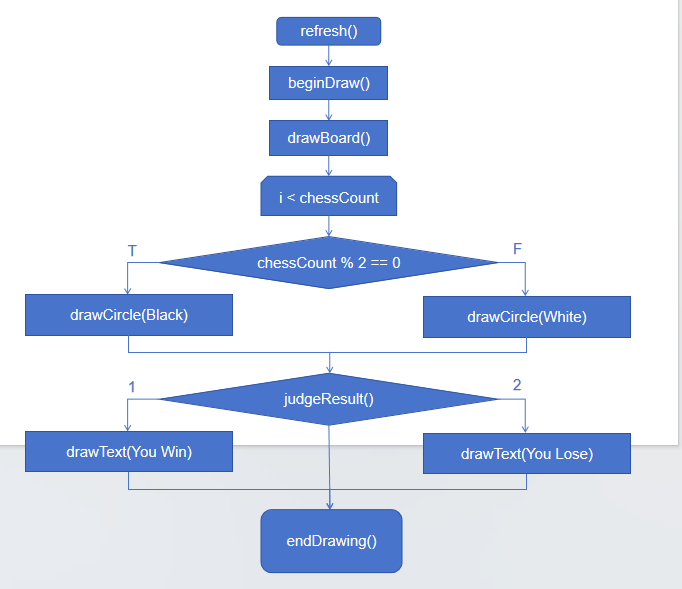
#### 5.三、四两点造成了测试的时候卡住不动或者跳过白棋下子的环节，单步调试修改以后五子棋就能基本正常进行了。

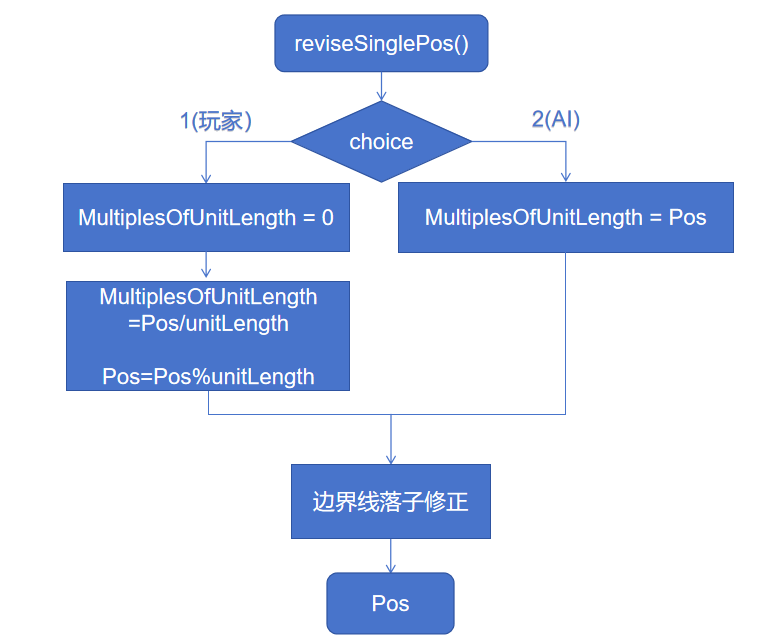
#### 6.有时候五连子的时候不会结束棋局，等到连第六个的时候才会。

#### 调试发现是因为棋型判别函数在数到第五个子时读取了第五个连子边的空白，导致估值函数没有返回五连子的值。在棋型判别函数循环中加上了if，判断到五连子（chessNum==5，space==0, block==0）情况时退出循环，直接调用judgeResult函数。

# 六、附图

（图一） （图二）

（图三）

（图四）

## alphaBetaPruning（图五）